

Creating an interactive Rating component for Mobile Web Templates

This article explains how to create and use an interactive Rating component for [Mobile Web Templates](#).

Introduction

Mobile Web Templates for High-End devices offer a range of [interactive and non-interactive components](#) that can be easily integrated in a mobile Web page or a Web Runtime widget. These components include a **static Rating, that enables the display of a 0- to 5-star rating.**



This article will show how to build a similar component, but with **fully interactive behavior**, so allowing users to freely express a rating, and **compatible with both touch-based and non-touch devices**.

Graceful degradation

In order to support also **browsers without JavaScript support**, a plain-HTML version of the component must be implemented. This guarantees that the **Rating component will work on all mobile browsers**, regardless of their scripting capabilities.

Designing the component

Let's start from the static Rating component offered by Mobile Web Templates.

The interactive component has to be **usable on both touch and non-touch devices**. This means that the touchable area must be **big enough to let the component be used with thumbs**, but must also **fit on displays of non-touch devices**, typically smaller than the one of touch ones.

Also, being the component interactive, it has to be able to **receive focus**. So, it is necessary to design **two states for the component**: with and without enabled focus.

These considerations done, a possible layout for the component is the following one:

UNFOCUSSED STATE



FOCUSSED STATE

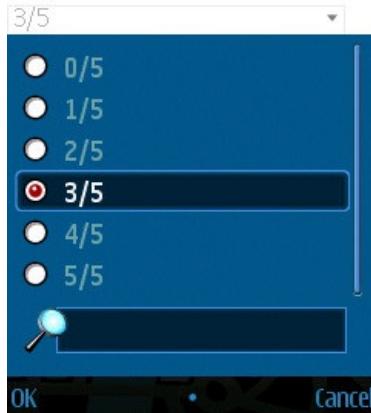


The plain-HTML version

On browsers without JavaScript support, an alternative, plain-HTML version must be used. In this case, since the Rating component allows users to choose in a range of 6 values (from 0 to 5), the most intuitive HTML element to use is a **SELECT input field**. The following picture shows how the plain-HTML version of the component appears on a S60 3rd edition device:

Voting Component

Select vote



Coding the component

The following sections show the coding process of the interactive Rating element.

The plain-HTML version

Let's start from the "**scaled-down**" version of the component, that will be implemented with an HTML SELECT element. This means that, in the Web page HTML code, a code similar to the following one must be inserted:

```
<label for="vote">Select vote</label>
<select name="vote" id="vote">
  <option value="0">0/5</option>
  <option value="1">1/5</option>
  <option value="2">2/5</option>
  <option value="3">3/5</option>
  <option value="4">4/5</option>
  <option value="5">5/5</option>
</select>
```

Starting from this SELECT element, the fully interactive component will be built.

Rating constructor

The component's constructor needs **2 arguments** to properly initialize the Rating component:

- **the DOM element**, defined above, representing the plain-HTML version
- **a starting value** to initialize the component

So, it is possible to define the following constructor:

```
function Rating(_id, _defaultValue)
{
  this.id = _id;
  this.ratingInput = document.getElementById(_id);
  this.value = (_defaultValue ? _defaultValue : 0);

  (this.ratingInput) ? this.init() : false;
}
```

The DOM structure

The dynamic component will have **2 main elements**:

- **the stars image**, showing a graphic representation of the current vote
- **a text element**, showing the textual representation of the current vote.

So, it is possible to define the following DOM structure for the component:

```
<!-- the root element -->
<ul class="voting">

  <!-- the actual component element -->
  <li class="voting-item voting-stars-5">

    <!-- an A element to support focus on 3rd edition devices -->
    <a href="#">
      <span></span>
    </a>
  </li>

  <!-- element showing the current rating value -->
  <li class="voting-value">
    5/5
  </li>

</ul>
```

CSS and images

The component is styled using the same approach used for the static Rating component. So, there will be **a single image used as background** of the rating element, that will be different for the focused and unfocused states:



and **separate CSS class for each different rating**, that will appropriately position the background image:

```
ul.voting li.voting-stars-0 {
  background-position: center 0px;
}
ul.voting li.voting-stars-1 {
  background-position: center -50px;
}
ul.voting li.voting-stars-2 {
  background-position: center -100px;
}
ul.voting li.voting-stars-3 {
  background-position: center -150px;
}
ul.voting li.voting-stars-4 {
```

```
background-position: center -200px;  
}  
ul.voting li.voting-stars-5 {  
    background-position: center -250px;  
}
```

The following section shows how this DOM structure will be dynamically built by the Rating component.

Component initialization

The component initialization has to perform these steps:

- **build the component's DOM structure**
- add the appropriate **event handlers** to the DOM elements
- initialize the component with the **default value**
- **hide the plain-HTML version** of the component

The following function performs these initialization steps:

```
Rating.prototype.init = function()  
{  
    //hide the text input  
    this.ratingInput.style.display = 'none';  
  
    //create the root <ul> element  
    var ul = document.createElement('ul');  
    ul.setAttribute('class', 'voting');  
  
    //create the rating <li> element  
    this.rating = document.createElement('li');  
  
    //create an inner A element  
    var aElement = document.createElement('a');  
    aElement.href = "#";  
  
    //create a SPAN for the A element  
    this.handle = document.createElement('span');  
  
    //create the rating value element  
    this.ratingValue = document.createElement('li');  
    this.ratingValue.setAttribute('class', 'voting-value');  
  
    //append elements  
    aElement.appendChild(this.handle);  
    this.rating.appendChild(aElement);  
    ul.appendChild(this.rating);  
    ul.appendChild(this.ratingValue);  
    this.ratingInput.parentNode.insertBefore(ul, this.ratingInput);  
  
    var self = this;  
  
    // event for key-based navigation  
    aElement.onkeydown = function(e)  
    {  
        self.handleKeyDown(e);  
    }  
  
    // event for touch devices
```

```
this.rating.onclick = function(e)
{
  self.handleClick(e);
}

// update the component with the initial value
this.update();
}
```

Component behavior

It is now necessary to define a function that allows to **update the value of the Rating component**. This function will actually just update the "value" property of the component:

```
Rating.prototype.setValue = function(_value)
{
  this.value = _value;

  this.update();
}
```

Then, it is necessary to define the update() function, that will **perform the UI changes to the component, to reflect its updated value**:

```
Rating.prototype.update = function()
{
  this.ratingInput.value = this.value;

  this.rating.setAttribute('class', 'voting-item voting-stars-' + this.value);

  this.ratingValue.innerHTML = this.value + "/5";
}
```

Handling touch events

Touch-based devices will allow users to interact with the component with thumbs/stylus based interactions. Since precise interactions are not always possible on mobile devices, the touch events will be handled in a "simplified" manner: **each touch on the component increases the rating value by one**, restarting from zero when it reaches the highest value (5).

This behavior is implemented by the following function:

```
Rating.prototype.handleClick = function(e)
{
  this.setValue((this.value + 1) % 6);
}
```

Handling key-based events

On **non-touch devices with tabbed navigation enabled**, the component has to intercept and respond to key events in the following manner:

- when the **left key** is pressed, the rating value is decreased by one
- when the **right key** is pressed, the rating value is increased by one

The following function implements this behavior:

```
Rating.prototype.handleKeyDown = function(e)
{
    var key = e.keyCode;

    var newValue = this.value;

    if(key == 37 && this.value > 0)
        newValue--;
    else if(key == 39 && this.value < 5)
        newValue++;

    if(newValue != this.value)
        this.setValue(newValue);
}
```

Using the component

The steps required to use the interactive Rating component are the following:

- **define the plain-HTML version** in the mobile Web page HTML code:

```
<select name="vote" id="vote">
    <option value="0">0/5</option>
    <option value="1">1/5</option>
    <option value="2">2/5</option>
    <option value="3">3/5</option>
    <option value="4">4/5</option>
    <option value="5">5/5</option>
</select>
```

- **call the component's constructor:**

```
var myRating = new Rating('vote', 2);
```

- **to retrieve the component's value**, just access its "value" property, or the value of the HTML SELECT element:

```
// accessing the component's value property

var myValue = myRating.value;

// accessing the SELECT value

var myValue = document.getElementById('vote').value;
```

Screenshots

The following screenshots show the component in action on **S60 5th edition devices**:

Voting Component

Select vote



Options

Exit

and on **S60 3rd edition FP2 devices**:

Voting Component

Select vote



Options

Exit

Downloads

You can download the following resources related to this article:

- The **interactive Rating component source code** (JavaScript, CSS, images):
<http://www.jappit.com/uploads/wrt/MWTRatingComponent.zip>
- A sample **Web Runtime widget** using the Rating component: <http://www.jappit.com/uploads/wrt/MWTRating.wgz>